

1     Improving Performance of Intermediate Nodes  
2                     with Flow Splicing

3     Field of The Invention

4     The invention is directed to communication via  
5     packet switched networks. It more particularly is  
6     directed to methods for improving the performance of  
7     intermediate network nodes that participate at or above  
8     the transport layer in communication between one or  
9     more node pairs.

10    Background of the Invention

11    Traditionally, internet software has been developed  
12    in accordance with the principle that very limited  
13    functionality is provided in "the network."  
14    Specifically the network provides unreliable forwarding  
15    of messages or packets. Following this model,  
16    additional functionality, such as reliable delivery and  
17    flow control, is implemented entirely at communication  
18    end points without adding any functionality to the  
19    network.

20    Figure 1 depicts a TCP connection between a first  
21    node 101 and a second node 102 following this model.  
22    The connection extends "directly" from the first node  
23    101 to the second node 102 with only IP routers 103 in  
24    between. Routers do not maintain state information  
25    associated with constructs that reside above the

1 network layer. Connection state, which is associated  
2 with the transport and session layer, is therefore  
3 maintained solely in the two connection end points,  
4 that is at the first node 101 and second node 102.

5 As internet technology has evolved, significant  
6 functionality has found its way into "the network."  
7 For example, it is increasingly common for a client  
8 running a web browser to connect to an *intermediate*  
9 node rather than directly to a web server. Such  
10 intermediate nodes include but are not limited to:  
11 SOCKS servers, fire walls, VPN (virtual private  
12 network) gateways, TCP routers or load balancers,  
13 caching proxies and transcoding proxies used with  
14 resource-constrained clients such as handheld devices.

15 The number and types of such intermediate nodes will  
16 continue to increase as internet technology evolves to  
17 provide additional functionality. The performance of  
18 the intermediate nodes will increase in importance as  
19 the number of internet clients continues to grow  
20 rapidly. This growth will be fueled by large numbers  
21 of handheld and pervasive devices that will require  
22 intermediate nodes capable of supporting hundreds of  
23 thousands to millions of clients simultaneously.

24 Figure 2 depicts this increasingly common scenario  
25 in which a first node 201 connects to a second node 202  
26 via a set of intermediate nodes 203. Although not  
27 explicitly depicted in the figure, intermediate nodes  
28 typically communicate with each other as well as with  
29 the first node and the second node via routers.

30 Each intermediate node influences communication  
31 between the first node 201 and the second node 202. In  
32 many instances, the interface provided by an

1 intermediate node to the first node 201 resembles or is  
 2 identical to the interface provided by the second node  
 3 202 to the intermediate node. Similarly, the interface  
 4 provided by an intermediate node to the second node 202  
 5 is typically similar to the interface provided by the  
 6 first node 201 to the intermediate node. Because of  
 7 this similarity of interfaces, the first node may not  
 8 be able to determine if is connected to an intermediate  
 9 node or to the second node. Similarly there may be no  
 10 way for the second node to determine if it is connected  
 11 to an intermediate node or to the first node. The  
 12 similarity of interfaces allows there to be any number  
 13 of intermediate nodes between the first node 201 and  
 14 the second node 202. This property is depicted in  
 15 Figure 3.

16 A key difference between a router and an  
 17 intermediate node is that routers perform processing  
 18 only at layers one through three of the ISO seven layer  
 19 model, those are the *physical*, *data link* and *network*  
 20 layers, while intermediate nodes perform processing at  
 21 and possibly above the fourth or *transport* layer.  
 22 Figure 4 depicts the processing performed by a router  
 23 in terms of layers. The corresponding diagram for  
 24 intermediate nodes is shown in Figure 5.

25 We define a *connection* to be a bi-directional  
 26 communication channel between a pair of connection end  
 27 points such that information written to one end of the  
 28 connection can be read from the other end. A  
 29 connection includes two *flows*. A flow is a  
 30 unidirectional communication channel between a pair of  
 31 flow end points such that information written to the  
 32 source flow end point can subsequently be read from the

1 corresponding destination flow end point. A connection  
2 need not necessarily be supported by a connection  
3 oriented protocol. All that is required is the  
4 identification of a pair of connection end points and  
5 propagation of data between the end points.  
6 Connections and flows reside at the fifth or *session*  
7 layer in the ISO model. Because routers in general do  
8 not perform processing above layer three, routers  
9 generally do not perform processing explicitly related  
10 to connections. Intermediate nodes however do  
11 generally perform processing explicitly associated with  
12 connections.

13 In fact, intermediate nodes can be distinguished  
14 from routers in terms of connections. In Figure 1, the  
15 first node 101 communicates with the second node 102  
16 via a single connection 104 whereas in Figure 2  
17 communication between the two nodes takes place via  
18 multiple connections in series 204, 205, 206, 207, 208  
19 and 209. The use of multiple connections provides each  
20 intermediate node with end points that can be used to  
21 influence communication between the first node 201 and  
22 the second node 202. However, intermediate nodes  
23 typically expend much of their resources simply moving  
24 data from one connection to another. In one common  
25 scenario, the intermediate node monitors the flow of  
26 information between the first node 201 and second node  
27 202 only until a request made by the first node 201 can  
28 be identified. In another common scenario, the  
29 intermediate node monitors the flow of information only  
30 in one direction. The performance and capacity of an  
31 intermediate node is often determined therefore by the

1 efficiency with which it moves data between  
2 connections.

3 Having a series of connections between the first and  
4 second node can cause several undesirable side-effects.  
5 Connections in series tends to deliver worse  
6 performance in terms of latency compared to a single  
7 connection and may also degrade throughput. Each node  
8 in a packet switched network introduces some delay and  
9 imposes a throughput limit. The performance of a  
10 packet switched network therefore relies on minimizing  
11 the delay introduced and maximizing the throughput  
12 supported by each node. This is accomplished, in part,  
13 by performing only minimal processing at each node.  
14 The packet forwarding performed by a router entails  
15 only a small amount of overhead, but the processing  
16 associated with connection end points performed by an  
17 intermediate node is significant.

18 The presence of multiple connections also alters the  
19 semantics of communication between the first node 201  
20 and the second node 202 of Figure 2. For example, with  
21 a single connection, the first node 201 is assured data  
22 has arrived at the second node 202 when it receives an  
23 acknowledgment. With multiple connections, the first  
24 node 201 may be led to believe data has arrived at the  
25 second node 202 when, in fact, it has only reached the  
26 first intermediate node.

## 27 Summary of the Invention

28 It is therefore an object of the present invention  
29 to improve the performance of intermediate nodes using

1 an approach called *flow splicing*. A flow splice  
 2 transforms a pair of flows, a first flow inbound to an  
 3 intermediate node and a second flow outbound from the  
 4 intermediate node, into a single third composite flow  
 5 that originates at the source of the first flow and  
 6 terminates at the destination of the second flow. The  
 7 transformation is invisible from the source node at  
 8 which the first flow originated and the destination  
 9 node at which the second flow terminated.

10 A flow splice is applied to a pair of flows in one  
 11 direction between a first and second node independent  
 12 of any associated flows. For example, a flow splice  
 13 transforms one flow of a connection without modifying  
 14 the other flow associated with the same connection.  
 15 The unidirectional nature of a flow splice is an  
 16 essential characteristic as techniques used to splice  
 17 connections do not generally allow data flow to be  
 18 spliced in one direction only. The ability to splice in  
 19 one direction only greatly increases the number of  
 20 scenarios in which splicing can be applied.

21 Figure 6 depicts a flow splice. In this figure, a  
 22 splice is performed at the intermediate node I 603  
 23 between the inbound flow from a first node 601 and the  
 24 outbound flow to a second node 602. Flow splicing  
 25 significantly improves network performance between  
 26 nodes communicating via one or more network  
 27 intermediaries and increase the capacity of the  
 28 intermediaries.

## 29 Brief Description of Drawings

1 The foregoing and other objects, aspects and  
2 advantages of the invention may be better understood by  
3 referring to the following detailed description of a  
4 preferred embodiment with reference to the accompanying  
5 drawings, in which:

6 Figure 1 is a diagram of two nodes communicating via  
7 routers;

8 Figure 2 is a diagram of two nodes communicating via  
9 intermediate nodes;

10 Figure 3 is a conceptual diagram that depicts  
11 interfaces exported and bound to by intermediate nodes;

12 Figure 4 is a diagram that depicts layer processing  
13 performed by routers;

14 Figure 5 is a diagram that depicts layer processing  
15 performed by intermediate nodes;

16 Figure 6 is a diagram that depicts flow  
17 transformation resulting from a flow splice in  
18 accordance with the present invention;

19 Figure 7 is a schematic diagram of two nodes  
20 communicating via a connection;

21 Figure 8 is a schematic diagram of two nodes each of  
22 which has established a connection with an intermediate  
23 node;

24 Figure 9 is a schematic diagram that depicts the  
25 effect of using a flow splice to transform two flows  
26 into a single flow in accordance with the present  
27 invention;

28 Figure 10 is a schematic diagram that depicts the  
29 effect of using a flow splice to transform two flows  
30 into a single flow in accordance with the present  
31 invention;

1 Figure 11 is a schematic diagram that depicts the  
2 effect of using a pair of flow splices to transform two  
3 connections into a single connection.

4 Figure 12 is a schematic diagram that depicts the  
5 effect of applying two splices to transform six flows  
6 into four in a scenario in which three nodes have each  
7 established a connection with a fourth node in  
8 accordance with the present invention;

9 Figure 13 is a diagram of TCP header format;

10 Figure 14 is a state diagram that shows various  
11 states for a TCP to TCP splice and the logic that  
12 governs state transitions in accordance with the  
13 present invention;

14 Figure 15 is a flow diagram that provides an  
15 overview of the logic for processing TCP segments  
16 associated with a flow splice in accordance with the  
17 present invention;

18 Figure 16 is a flow diagram that describes splice  
19 destination processing in accordance with the present  
20 invention;

21 Figure 17 is a flow diagram that describes splice  
22 source processing in accordance with the present  
23 invention;

24 Figure 18 is a graphical representation of a  
25 sequence of packets flowing through an intermediate  
26 node without flow splicing;

27 Figure 19 is a graphical representation of a  
28 sequence of packets flowing through an intermediate  
29 node with a flow splice in accordance with the present  
30 invention.



# 1 Detailed Description of the Invention

## 2 Splice Architecture

3 A node is any device capable of communicating via  
 4 one or more communication networks or links. Figure 7  
 5 depicts two nodes A 701 and B 702 communicating via a  
 6 connection  $C_{AB}$  705. The connection  $C_{AB}$  705 is  
 7 bi-directional and includes of two flows. Flow  $F_{AB}$  703  
 8 carries data from node A 701 to node B 702. Similarly,  
 9 data flows in the reverse direction, that is from node  
 10 B 702 to node A 701 along flow  $F_{BA}$  704. A connection  
 11 therefore includes of two flows, one in each direction,  
 12 between a pair of nodes.

13 A connection may also be denoted by a pair of end  
 14 point pairs. In Figure 7, connection  $C_{AB}$  705 extends  
 15 between end point pair  $EP_{A1}$  706 at node A 701 and end  
 16 point pair  $EP_{B1}$  707 at node B 702. End point  $EP_{A1}$  706  
 17 denotes the flow outbound from node A 701 via  $F_{AB}$  703  
 18 and the inbound flow to node A 701 via  $F_{BA}$  704.  
 19 Similarly, end point pair  $EP_{B1}$  707 denotes the flow  
 20 inbound to node B 702 via  $F_{AB}$  703 and the flow outbound  
 21 from node B 702 via  $F_{BA}$  704. The order in which the  
 22 nodes linked by a connection are listed in the  
 23 connection name is not significant. For example the  
 24 name  $C_{BA}$  denotes exactly the same connection as  $C_{AB}$  705.

25 The nature of a flow is that data written to the  
 26 source flow end point can subsequently be read from the  
 27 destination flow end point. Although boths ends of a  
 28 flow can reside on the same node, flows are commonly  
 29 extended between nodes through the use of transport  
 30 protocols such as TCP/IP, AppleTalk and NETBIOS.

1 One aspect of the present invention allows two  
2 flows, one inbound the other outbound relative to a  
3 given node, both flows being terminated at the given  
4 node, to be transformed into a single flow through the  
5 given node. The method of the present invention  
6 transforms the two flows into one flow which originates  
7 at the source of the original inbound flow and  
8 terminates at the destination of the original outbound  
9 flow. The method does not require any particular  
10 relationship between the two flows to be transformed  
11 other than that one flow be inbound the other flow be  
12 outbound at the intermediate node at which the  
13 transformation is performed. The two flows may or may  
14 not be associated with the same connection. The source  
15 of the inbound flow and destination of the outbound  
16 flow may reside on the same node or on different nodes.  
17 The term "flow splice" or simply "splice" is used  
18 herein to refer to this described transformation.

19 Before two flows are spliced, any amount of data can  
20 be read from the inbound flow and written to the  
21 outbound flow. Subsequent to the creation of a splice,  
22 data arriving on the inbound flow is automatically sent  
23 on the outbound flow. That is, data is propagated to  
24 the outbound flow without any further action by the  
25 entity that invoked the splice operation. Except for  
26 performance considerations, a pair of spliced flows  
27 resembles a pair of flows where data is read from the  
28 inbound flow as it arrives and is immediately written  
29 to the outbound flow.

30 The splice operation has a wide variety of  
31 applications. For example the connections depicted in  
32 Figure 8 could correspond to any of many different

1 transport protocols including IP, UDP, TCP, IPX, OSI,  
2 NETBIOS, AppleTalk, etc. Any of several application  
3 programming interfaces (APIs) could be used to create  
4 and manipulate the flows transformed by a splice.  
5 These include the Berkeley Socket Interface, the  
6 STREAMS interface, NETBEUI and IBM's Common Programming  
7 Interface for Communication (CPI-C). Both flows  
8 transformed by a given splice may be associated with  
9 the same transport protocol or each may be associated  
10 with a different protocol. Each protocol may be either  
11 connection oriented or connection less. We proceed to  
12 describe the method of the present invention in its  
13 most general form and subsequently describe specific  
14 instances of the invention's application in more  
15 detail.

16 The specific details of implementing the splice  
17 operation vary somewhat depending on the particular  
18 transport protocols involved. One characteristic  
19 feature of a splice implementations is however that it  
20 "short circuits" or reduces protocol processing at and  
21 above the transport layer. An obvious approach to  
22 reducing processing above the transport layer is for  
23 the splice implementation to simply modify each packet  
24 received on the inbound flow to make it suitable for  
25 transmission on the outbound flow and immediately send  
26 the packet.

27 Another aspect of the present invention defines an  
28 architecture for manipulating data flows that can be  
29 combined with an existing communication architecture.  
30 The only requirement of the existing architecture is  
31 that it allows the establishment of data flows.

1 Figure 8 depicts a scenario that includes three  
2 nodes A 801, I 802, and B 803 and two connections C<sub>AI</sub>  
3 808 and C<sub>BI</sub> 809. Node I 802 represents an *intermediate*  
4 node between nodes A 801 and B 803. An intermediate  
5 node is a node that influences communication between  
6 one or more pairs of nodes that communicate with each  
7 another by connecting to the intermediate node. There  
8 are many different ways in which an intermediate node  
9 can influence communication between node pairs. For  
10 example, an intermediate node might simply enable  
11 communication between two sets of nodes that would not  
12 otherwise be able to communicate. An intermediate node  
13 might enforce security restrictions by allowing each  
14 node to communicate with only a specifically authorized  
15 set of nodes. An intermediate node might also improve  
16 the performance of communication between two sets of  
17 nodes by *caching*, that is retaining a copy of, data  
18 transferred between the sets of nodes and providing  
19 such data to a node when it is requested eliminating  
20 the need for the data to be resent by the node from  
21 which it was originally obtained.

22 Figure 9 shows the scenario that results from  
23 transforming the scenario in Figure 8 using a flow  
24 splice. Specifically, Figure 9 shows the results of  
25 transforming Figure 8 by splicing flow F<sub>AI</sub> 804 to flow  
26 F<sub>IB</sub> 806 at node I 802. The splice transforms the two  
27 flows, F<sub>AI</sub> 804 and F<sub>IB</sub> 806, into a single flow F<sub>AB</sub> 904.  
28 This splice operation can be described in terms of end  
29 point pairs at node I 802. Specifically, Figure 9  
30 depicts the inbound flow associated with end point pair  
31 EP<sub>I1</sub> 811 having been spliced to the outbound flow  
32 associated with end point pair EP<sub>I2</sub> 812. A splice

1 operation is therefore performed on a pair of flows.  
2 One flow is inbound and the other flow is outbound  
3 relative to the intermediate node at which the splice  
4 is performed.

5 By combining two flows into one, the splice  
6 eliminates two flow end points. The two are the  
7 destination end point of the inbound flow, and the  
8 source end point of the outbound flow. Note two of the  
9 so-called end point pairs in Figure 9, specifically  
10  $EP_{I1}$  908 and  $EP_{I2}$  909 include only a single end point.  
11 Also note Figure 9 does not show any connections as  
12 previously defined. That is it does not show any flow  
13 pairs in opposite directions between the same pair of  
14 nodes. The splice depicted in Figure 9 therefore  
15 eliminates the connections  $C_{A1}$  808 and  $C_{B1}$  809 shown in  
16 Figure 8.

17 The splice depicted in Figure 9 changes the  
18 destination of the outbound flow associated with end  
19 point pair  $EP_{A1}$  810 on node A 801 and similarly changes  
20 the source of the inbound flow associated with end  
21 point pair  $EP_{B1}$  813 on node B 803. Despite this, the  
22 splice is transparent to both node A 901 and node B  
23 903. Nothing intrinsic to the splice allows an  
24 observer on node A 901 to detect that data sent  
25 outbound from end point pair  $EP_{A1}$  907 ends up at node B  
26 903 instead of node I 902. Similarly, nothing  
27 intrinsic to the splice allows an observer on node B to  
28 detect that data received on end point pair  $EP_{B1}$  910  
29 originates at node A 901 instead of node I 902.

30 The splice is, in fact, undetectable from node B 903  
31 unless either the data received on the inbound flow  
32 associated with end point pair  $EP_{B1}$  910 or the rate at

1 which it arrives identifies the sender. Similarly the  
2 splice can not be detected from node A 901 unless the  
3 rate at which data is received after being sent on the  
4 outbound flow associated with end point pair  $EP_{A1}$  907  
5 can be detected at node A 901 in such a way as to  
6 identify the receiver.

7 Figure 10 also depicts a scenario that results from  
8 transforming the scenario in Figure 8 with a flow  
9 splice. Figure 10 shows the result of splicing the  
10 inbound flow associated with end point pair  $EP_{I2}$  812  
11 with the outbound flow of end point pair  $EP_{O1}$  811.  
12 Splicing these two flows results in a single flow  $F_{BA}$   
13 1006 shown in Figure 10. Figures 9 and 10 together  
14 show that the flows *in either direction* between a pair  
15 of nodes communicating via an intermediate node can be  
16 transformed into a single flow via a splice.

17 Figure 11 shows the scenario that results from  
18 transforming the scenario in Figure 8 with two splices,  
19 those depicted in Figure 9 and Figure 10. One splice  
20 transforms flows  $F_{AI}$  804 and  $F_{IB}$  806 into flow  $F_{AB}$  1104.  
21 The other splice transforms flows  $F_{BI}$  807 and  $F_{IA}$  805  
22 into flow  $F_{BA}$  1105. Unlike the scenarios depicted in  
23 Figure 9 and Figure 10, the scenario in Figure 11  
24 includes a connection. The two splices transform  
25 connections  $C_{AI}$  808 and  $C_{BI}$  809 into connection  $C_{AB}$  1106.  
26 Thus just as a flow splice transforms two flows into  
27 one, a pair of flow splices can transform two  
28 connections into one.

29 More complicated examples of flow splicing are  
30 possible. Figure 12 shows an example with four nodes,  
31 A 1201, I 1202, B 1203, and C 1204 and four flows  $F_{AI}$   
32 1205,  $F_{CA}$  1206,  $F_{IB}$  1207 and  $F_{BC}$  1208. The scenario in

1 Figure 12 resulted from transforming a scenario with  
 2 three connections, one from node I 1202 to each of  
 3 nodes A 1201, B 1203, and C 1204 with two flow splices.  
 4 Another likely scenario is the case where a series of  
 5 flows from a source node to a destination through a set  
 6 of network intermediaries are transformed into a single  
 7 flow from the source to destination using a number of  
 8 flow splices.

9 We now consider the use of flow splicing. The  
 10 splice architecture allows a flow to be spliced after  
 11 any amount of data, or no data, has been sent or  
 12 received on the flow. The architecture also allows a  
 13 spliced flow to revert to the unspliced state either  
 14 due to termination of the inbound flow or because the  
 15 splice was explicitly eliminated at the intermediate  
 16 node. This flexibility allows flow splicing to be used  
 17 in a variety of ways.

18 One potential use for flow splicing is *load*  
 19 *balancing*. An intermediate node can use flow splicing  
 20 to balance the load from a set of clients across a set  
 21 of servers. An intermediate node can implement load  
 22 balancing by establishing one or more flows with client  
 23 nodes, establishing one or more flows with server nodes  
 24 and splicing flows between the clients and servers.

25 Another use for flow splicing is *content*  
 26 *partitioning*. This refers to dividing a set of content  
 27 (or services) across a set of servers possibly with  
 28 multiple servers providing the same content. An  
 29 intermediate node can make it appear to clients that  
 30 all the content is available from a single server node.  
 31 A content partitioning intermediate node operates

1 similarly to a load balancing intermediate node in  
2 terms of splicing flows between clients and servers.

3 As an example we consider an intermediate node that  
4 allows World Wide Web content to be partitioned across  
5 a set of servers. The web intermediate node starts by  
6 accepting a TCP connection from a client. It then  
7 reads one or more requests from the flow inbound from  
8 the client. Next the intermediate node determines an  
9 appropriate server to service the client request(s).  
10 It then establishes a connection, including of two  
11 flows one in either direction, with the selected server  
12 assuming an appropriate connection is not already  
13 available. The intermediate node then selects the  
14 inbound flow from the chosen server and the outbound  
15 flow to the client and splices the selected flows.  
16 Finally, the intermediate node writes the request that  
17 it had received from the client to the server.

18 The response from the server subsequently flows to  
19 the client via the splice. The client may close the  
20 connection after sending a request in which case  
21 processing for the connection is complete. The client  
22 might also send another request on the same connection  
23 in which case the intermediate node can either write  
24 the request to the same server allowing the response to  
25 flow across the established splice or resplice the  
26 outbound flow to the client to a different flow inbound  
27 from a server and send the client request on the  
28 corresponding flow outbound to that server instead.

29 It will be understood by those skilled in the art  
30 that the process just described can be applied to a  
31 wide variety of intermediate node functions and  
32 communication protocols. For example, the steps listed



1 above for the web intermediate node can be used to  
2 provide load balancing and content partitioning for a  
3 list of protocols including HTTP, SOCKS, telnet, FTP,  
4 AFS, DFS, NFS, RFS, SMTP, POP, DNS, Sun RPC, and NNTP.

5 One significant aspect of many applications of flow  
6 splicing is that the intermediate node may decide what  
7 node to establish either the first or second connection  
8 with based on several factors. For example the  
9 intermediate node may determine what node to establish  
10 the second connection with after parsing a request read  
11 from the first connection. The intermediate node might  
12 also determine what specific port or address within the  
13 first node to establish the first connection with based  
14 on the remote address associated with the second  
15 connection. The intermediate node might do this to  
16 direct requests from certain clients to certain  
17 servers. Other factors the intermediate node may  
18 consider include estimates of bandwidth and latency for  
19 the various flows, quality of service (QoS)  
20 requirements for the various flows, the number of  
21 network hops between any of the nodes, and  
22 configuration or state information stored at the  
23 intermediate node.

24 In addition to determining what address a connection  
25 should be established with, the intermediate node may  
26 determine whether to perform a splice at all based on  
27 several factors. For example an intermediate node may  
28 parse requests sent by a client to a server in order to  
29 estimate the amount of data expected to flow from the  
30 server to the client and splice only flows expected to  
31 handle a lot of data. The intermediate node might also  
32 splice only flows expected to support high bandwidth

1 and/or low latency. Finally the intermediate node might  
2 splice only those flows associated with a specific set  
3 of clients or servers.

4 We now describe a method that embodies the splice  
5 operation for the specific case where both flows joined  
6 by the splice are associated with TCP connections. It  
7 will be understood by those skilled in the art that the  
8 concept of transforming two flows, one inbound the  
9 other outbound at a given point, into a single flow  
10 from the source of the original inbound flow to the  
11 destination of the original outbound flow by suitably  
12 modifying packet headers can be applied to any packet  
13 based communication protocol.

14 Let us now consider TCP state. Every TCP end point  
15 pair has a set of associated state information. This  
16 information is referred to in the following discussion  
17 by the name TCB which stands for TCP Control Block.  
18 The specific way in which the state information is  
19 stored in a TCB is not important to the following  
20 discussion. The discussion does however refer to some  
21 specific elements of the TCB which are described as  
22 follows.

23 snd\_nxt - The sequence number of the next byte to be  
24 sent. One greater than the greatest sequence number  
25 sent so far.

26 snd\_una - The smallest sequence number associated  
27 with any byte that has been sent, but has not yet been  
28 acknowledged. Equal to the greatest acknowledgment  
29 value received.

30 rcv\_nxt - The sequence number of the next byte of  
31 data expected to arrive. One more than the greatest  
32 sequence number received.

1     rcv\_wnd - An offset from rcv\_nxt that identifies the  
2 largest sequence number a receiver is willing to  
3 receive.

4     Now we consider forward and reverse information.  
5 Figure 13 shows the structure of a TCP packet header.  
6 The information contained in a TCP packet may be  
7 divided into two categories. The first we call *forward*  
8 *outbound* information. This includes of the packet's  
9 data payload and associated sequence number 1303,  
10 checksum 1313, urgent pointer 1314, and any time stamp  
11 option value. The second category we call *reverse*  
12 *inbound* information. This includes the acknowledgment  
13 flag 1307 and field 1304, window size 1312, and any  
14 time stamp option echo value. The fields of the TCP  
15 header that contain reverse inbound information are  
16 shaded in Figure 13.

17     Certain fields of the TCP header and in the  
18 associated IP header identify the connection with which  
19 the segment is associated. These fields are the source  
20 and destination port number in the TCP header and the  
21 source and destination IP address in the IP header.  
22 Because these fields identify the connection, they are  
23 associated with both the forward outbound and reverse  
24 inbound information.

25     The forward information is intended for the end  
26 point pair to which data is sent on a TCP connection.  
27 The reverse information is intended for the end point  
28 pair from which data is received on the connection.  
29 Without flow splicing, these two end point pairs are  
30 identical. A splice, however, can break this equality.  
31 The ability to splice each flow associated with an end  
32 point pair independently relies on independent

1 processing of the forward outbound and reverse inbound  
2 information. The specific details of processing  
3 forward and reverse information in TCP packets is  
4 described in further detail below.

5 The next aspect to consider is splice states. A  
6 flow splice joins two flows into one thus eliminating a  
7 pair of flow end points. Several factors may make it  
8 impossible for the end points to be eliminated  
9 immediately when a splice is created. A splice  
10 therefore has an associated state value that influences  
11 its operation. We now discuss various factors that  
12 influence a splice's state and the operation of a  
13 splice in each state. A state diagram showing the  
14 various splice states and the logic for transitions  
15 between them is shown in Figure 14.

16 One consideration that can prevent spliced end  
17 points from being eliminated is the presence of data in  
18 send or receive buffers associated with the spliced  
19 flows. Each flow end point generally has an associated  
20 buffer. An inbound flow has a receive buffer that  
21 contains data that has been received but has not yet  
22 been read by the application. An outbound flow has a  
23 send buffer that contains data that has been written by  
24 the application but has not yet been successfully sent.  
25 Any data that resides in the send buffer when the  
26 splice is created is sent before packets can be  
27 forwarded via the splice. Similarly any data in the  
28 receive buffer is sent after any data in the send  
29 buffer and before any packets are forwarded.

30 If data resides in either the send or receive buffer  
31 when a splice is created, the splice starts in the  
32 PENDING 1402 state. Any data residing in the receive

1 buffer when the splice is created is appended to the  
 2 contents, if any, of the send buffer. The receive  
 3 buffer is then essentially eliminated as any data  
 4 subsequently arriving at a splice in the PENDING 1402  
 5 state is appended to the send buffer associated with  
 6 the outbound flow instead of being appended to the  
 7 receive buffer of the inbound flow as would be the case  
 8 for data arriving on an unspliced flow.

9 Full protocol processing is performed for each end  
 10 point associated with a splice in the PENDING 1402  
 11 state. A PENDING 1402 splice behaves similarly to a  
 12 program that reads data from the inbound flow as it  
 13 arrives and writes it to the outbound flow. The splice  
 14 however eliminates the need to invoke a separate  
 15 program, possibly saving protection domain and context  
 16 switches, and also eliminates any data copies between  
 17 the protocol implementation and the program.

18 A splice remains in the PENDING 1402 state at least  
 19 until its send buffer is empty. Additional data  
 20 arriving at a PENDING 1402 splice may thwart attempts  
 21 to empty the buffer. A PENDING 1402 splice may  
 22 therefore limit the amount of data it receives by  
 23 closing its *receive window*. The receive window is an  
 24 item of reverse inbound information that informs a  
 25 sender how much data a receiver is willing to receive.  
 26 The TCP protocol forbids a receiver from decreasing its  
 27 receive window by an amount greater than the amount of  
 28 data it has received. In other words, once a receiver  
 29 has indicated it is able to receive a given amount of  
 30 data, it honors its commitment. However, if a receiver  
 31 indicated it is capable of receiving X bytes of data

1 and subsequently receives Y bytes it is permitted to  
2 decrease its receive window to  $X - Y$ .

3 A PENDING TCP flow splice could also limit the  
4 amount of data it receives by withholding  
5 acknowledgments for data that it receives. The lack of  
6 acknowledgments will discourage the sender from sending  
7 additional data. Withholding acknowledgments may also  
8 cause the sender to retransmit data that has already  
9 been sent. For this reason it may be advantageous for  
10 a PENDING splice to limit the amount of data it  
11 receives by decreasing the send window rather than by  
12 withholding acknowledgments.

13 Regardless of the presence of buffered data, a  
14 splice is also prevented from proceeding beyond the  
15 PENDING 1402 state if the receive window advertised to  
16 the splice source (by the splice) is greater than the  
17 receive window advertised by the splice destination (to  
18 the splice). If this situation arises, the splice  
19 remains PENDING 1402 until it has received enough data  
20 to allow it to eliminate the gap between the receive  
21 window sizes.

22 Once the last byte of buffered data if any has been  
23 sent and the receive window has been decreased to the  
24 value advertised by the receiver if need be, the splice  
25 enters the FORWARDING 1403 state. In this state  
26 packets arriving at the splice are modified and  
27 forwarded. However, reverse inbound information  
28 associated with buffered data, as opposed to data  
29 simply forwarded through the splice, is handled in the  
30 same way as for an unspliced flow, rather than being  
31 forwarded to the sender. Once the splice is in the  
32 FORWARDING 1403 state, full protocol processing need

1 not be performed for the inbound flow. Processing of  
2 reverse inbound information, including performing  
3 retransmission still occurs, however, for the outbound  
4 flow.

5 Eventually the last reverse inbound information that  
6 refers to buffered data should be received at which  
7 point the splice enters the ESTABLISHED 1404 state. In  
8 this state both forward and reverse inbound information  
9 is processed (modified) and forwarded.

10 We now consider flow termination. When a node  
11 communicating via a splice terminates its flow, the  
12 termination can be handled by the splice in either of  
13 two ways. In one embodiment the splice propagates the  
14 termination, or *FIN* in TCP parlance just as it  
15 propagates ordinary data. This causes both flows  
16 joined by the splice to be shut down. In another  
17 embodiment the splice itself processes the *FIN*. This  
18 results in only the inbound flow at the splice being  
19 shut down and causes the outbound flow to revert to the  
20 UNSPLICED 1401 state. Having the splice propagate the  
21 *FIN* is desirable in situations where an intermediate  
22 node does not need to send any more data on the  
23 outbound flow after creating a splice. Having the  
24 splice process the *FIN* allows an intermediate node to  
25 either send additional data on the outbound flow, or  
26 ressplice the flow.

27 Once a splice has processed a *FIN* and a  
28 corresponding *ACK*, it enters the WAITING 1405 state.  
29 This state is identical to the ESTABLISHED 1404 state  
30 except for the presence of a timer that causes the  
31 splice to cease to exist when it expires. The timer is  
32 set to  $2 \times \text{MSL}$ , twice the maximum segment lifetime, when

1 the splice enters the WAITING 1405 state and reset to  
2 2\*MSL whenever a packet is forwarded through the  
3 splice. Usually no packets are forwarded through the  
4 splice once it enters the WAITING 1405 state and the  
5 splice is eliminated 2\*MSL after the FIN and  
6 corresponding ACK are processed. The purpose of the  
7 WAITING state is to account for any retransmitted  
8 packets.

9 The 2\*MSL time-out is conservative in that the  
10 splice persists only until it can be determined that  
11 any retransmissions that might occur would have been  
12 seen at the splice. However, the retransmission  
13 time-out value used by a TCP sender is a function of  
14 both the estimated round trip time and its variance and  
15 it is difficult for the splice to accurately estimate  
16 these values as viewed by the sender.

17 Now consideration is given to sequence and  
18 acknowledgment number mapping. TCP is a stream  
19 oriented protocol. Although a sending client presents  
20 data to the protocol in discrete buffers and the  
21 protocol partitions transmitted data into packets,  
22 neither buffer nor packet boundaries are visible to the  
23 receiver. TCP presents data to a receiver as a  
24 continuous stream bounded only by a single pair of  
25 beginning and end points. To ensure data is presented  
26 to the receiver in exactly the same order in which it  
27 was sent, TCP assigns a 32-bit sequence number to each  
28 byte in the stream. The *initial sequence number*, that  
29 is the sequence number for the first byte to traverse a  
30 flow, is generally selected in a way that makes it  
31 difficult to predict and the sequence number is  
32 incremented by one for each successive byte.



1 Because different flows generally have different  
2 initial sequence numbers and because any amount of data  
3 can be sent on a flow before it is spliced, the  
4 sequence numbers for data flowing through a splice are  
5 mapped from the inbound to the outbound flow. For  
6 example, consider two flows shown in Figure 8,  $F_{AI}$  804  
7 which extends from node A 801 to node I 802, and  $F_{IB}$   
8 806 that extend from the node I 802 to node B 803.  
9 Assume the initial sequence number for  $F_{AI}$  804 (chosen  
10 by node A 801) is 1,000, and the initial sequence  
11 number for  $F_{IB}$  806 (chosen by the node I 802) is  
12 10,000. Further assume 200 bytes has arrived at I 802  
13 on  $F_{AI}$  804 and 300 bytes have been sent from I 802 on  
14  $F_{IB}$  806 before  $F_{AI}$  804 is spliced to  $F_{IB}$  806. When the  
15 splice is created, the sequence number of the next byte  
16 to be received on  $F_{AI}$  804 is 1,200 and the sequence  
17 number of the next byte to be sent on  $F_{IB}$  806 is  
18 10,300. Thus as data flows through the established  
19 splice the sequence number in each TCP packet is mapped  
20 from the inbound flow,  $F_{AI}$  804, to the outbound flow  $F_{IB}$   
21 806 by adding 9,100 ( $10,300 - 1,200$ ). Acknowledgment  
22 numbers flowing in the opposite direction are mapped by  
23 subtracting 9,100.

24 Creating a splice between two sequence-oriented  
25 flows establishes a correspondence between the sequence  
26 space of the first flow and the sequence space of the  
27 second flow. In the case of a splice between two TCP  
28 flows, the correspondence amounts to a fixed offset  
29 between the sequence number of a byte arriving at the  
30 intermediate node on the inbound flow and the sequence  
31 number of the corresponding byte sent by the  
32 intermediate node on the outbound flow.

1 The correspondence established when a splice is  
2 created can be calculated in a number of ways. For  
3 example, if when a splice is created, no data resides  
4 in either send or receive buffers associated with the  
5 flows to be spliced at the intermediate node, the  
6 offset can be calculated by subtracting the sequence  
7 number of the next byte expected from the source  
8 (rcv\_nxt) from the sequence number of the next byte to  
9 be sent to the destination (snd\_nxt). If either the  
10 receive buffer associated with the inbound flow or the  
11 send buffer associated with the outbound flow contains  
12 data, the sequence number of the next byte to be sent  
13 on the outbound flow is adjusted to account for the  
14 buffered data that will be sent before any data is  
15 forwarded through the splice. Alternately, the  
16 calculation of the offset can be delayed until both the  
17 send and receive buffer are empty.

18 We refer to the inbound and outbound flows at the  
19 splice even though the splice transforms the two flows  
20 into one. Strictly speaking, we are referring to the  
21 sequence number space associated with the former flow  
22 from the splice source to the intermediate node and the  
23 sequence number space associated with the former flow  
24 from the intermediate node to the splice destination.  
25 For brevity we simply refer to the inbound and outbound  
26 flows.

27 Processing for a TCP to TCP Splice in the  
28 established state is now considered. Figure 15 is a  
29 flow diagram that gives an overview of the logic for  
30 processing TCP segments associated with a flow splice  
31 in the ESTABLISHED 1404 state. The initial processing  
32 is the same regardless of whether or not the segment is

1 associated with a splice. First, in block 1502,  
2 several preliminary checks are performed to make  
3 certain the packet contains a valid TCP segment. The  
4 state information for the end point pair with which the  
5 segment is associated is then located. If no such  
6 state information is found, the segment is discarded  
7 and processing for the segment is complete. If the  
8 outbound flow of the end point pair on which the  
9 segment arrived is the destination of a splice the  
10 reverse inbound information contained in the segment is  
11 subjected to splice destination processing. This  
12 processing is described in further detail below and is  
13 depicted in Figure 16.

14 If a segment is subject to splice destination  
15 processing and contains neither data nor a FIN, then  
16 processing for the segment is complete. If a segment  
17 does contain either data or a FIN or the segment was  
18 not subject to splice destination processing the  
19 inbound flow of the end point pair on which the segment  
20 arrived is checked in decision block 1506 to see if it  
21 is a splice source. If the flow is not a splice  
22 source, the segment is subjected to the remainder of  
23 ordinary TCP input processing. If the inbound flow is  
24 a splice source, the corresponding outbound flow is  
25 checked in decision block 1507 to determine if it is  
26 not the destination of a splice and the segment is  
27 checked to see if it contains new reverse inbound  
28 information. If these two conditions are true, a copy  
29 of the reverse inbound information in the segment is  
30 made in block 1508 so it can be subjected to the  
31 remainder of the ordinary TCP input processing. A  
32 segment is considered to contain new reverse inbound

1 information if it contains an acknowledgment number  
2 greater than the largest acknowledgment number  
3 previously seen on this flow, or if it contains a  
4 window update. The segment is next subjected to splice  
5 source processing which is described in further detail  
6 below and depicted in Figure 17. If a copy of the  
7 segment's reverse inbound information was made, that  
8 information is subjected to the remainder of ordinary  
9 TCP input processing in block 1511. This 1512  
10 completes processing for a TCP segment associated with  
11 a splice.

12 Figure 16 is a flow diagram that depicts splice  
13 destination processing. Processing begins in block  
14 1601 and in 1602 the acknowledgment number in the  
15 segment is mapped from the destination flow to the  
16 source flow, and the window value in the segment is  
17 unscaled and limited to the maximum value that can be  
18 communicated to the splice source. The limiting factor  
19 is required in cases where the node on which the splice  
20 is performed has negotiated a larger window scale value  
21 with the splice destination than with the splice  
22 source. The mapped acknowledgment number is then  
23 checked to determine if it is greater than the greatest  
24 acknowledgment number sent from the splice to the  
25 splice source (stored in the rcv\_nxt field of the  
26 source TCB) 1603. If so the mapped acknowledgment  
27 number is recorded 1604 in the rcv\_nxt field of the  
28 source TCB and processing continues with formulating a  
29 TCP segment containing reverse inbound information to  
30 be sent to the splice source in block 1606. Block 1606  
31 includes:

32 1. Allocate packet for TCP segment,

1     2. Fill in fields of IP header needed for TCP  
2 checksum (packet length, protocol, source, destination  
3 IP address),  
4     3. Fill in fields of TCP header (source, destination  
5 port, TCP header length, sequence and acknowledgment  
6 numbers, flags).  
7     If the mapped acknowledgment value is not greater  
8 than the rcv\_nxt field of the source TCB, the segment  
9 is checked to determine if it contains any data or a  
10 FIN 1605. If the segment contains either data or a FIN  
11 processing for the segment is complete. If not,  
12 processing continues with formulating a TCP segment  
13 containing reverse inbound information in block 1606.  
14     Formulating the TCP segment begins with allocating a  
15 new packet and filling in the fields of the IP pseudo  
16 header needed for the IP checksum, specifically the  
17 packet length, protocol, and source and destination IP  
18 address. The IP addresses are copied from the source  
19 TCB. Next the fields of the TCP header are filled in.  
20 These are the source and destination port numbers,  
21 sequence and acknowledgment numbers, TCP header length  
22 and flags. The port numbers are copied from the source  
23 TCB. The only flag set is ACK. The sequence number is  
24 copied from the snd\_nxt field of the source TCB. The  
25 acknowledgment field is set to the mapped  
26 acknowledgment value.  
27     A determination is made to see if the segment  
28 contains a window update 1607. If it does, the value  
29 for the receive window, and sequence and acknowledgment  
30 numbers are recorded in the destination TCB 1608. Then  
31 block 1609 is performed and the receive window field of  
32 the TCP header is copied from the destination TCB. The  
33 TCP checksum is calculated and filled in. The total  
34 length, type of service and time to live fields of the

1 IP header are filled in. Finally the packet is passed  
2 to the IP output routine and the process ends 1610.

3 In the embodiment described new reverse inbound  
4 information is propagated to the splice source  
5 immediately. The sending of reverse inbound  
6 information could also be delayed for a short period of  
7 time in the hope that it could be sent in segments  
8 containing forward data for the source instead of  
9 sending a packet containing only reverse inbound  
10 information.

11 Figure 17 is a flow diagram that depicts the steps  
12 of an example of splice source processing. The  
13 majority of processing is performed in block 1702 and  
14 includes modifying information in the TCP and IP packet  
15 headers. Step 1702 includes:

- 16 1. Modify source and destination address in IP
- 17 header
- 18 2. Modify source and destination port in TCP header
- 19 3. Map sequence number from source to destination
- 20 flow
- 21 4. Set acknowledgment field from rcv\_nxt field of
- 22 destination TCB
- 23 5. Set window field from rcv\_wnd field of
- 24 destination TCB
- 25 6. Set length field in IP pseudo header
- 26 7. Calculate and fill in TCP checksum
- 27 8. Fill in IP total length, type of service and time
- 28 to live fields

29 The source and destination IP addresses and port  
30 numbers are modified by copying the appropriate values  
31 from the destination TCB. The sequence number is  
32 mapped from the source flow to the destination flow.

1 The acknowledgment number and receive window fields are  
2 set by copying the values from the rcv\_nxt and rcv\_wnd  
3 fields of the destination TCB respectively. The length  
4 field in the IP pseudo header is set. The TCP checksum  
5 is calculated and filled in. The total length, type of  
6 service, and time to live fields of the IP header are  
7 filled in.

8 If the segment's FIN flag is set TCP state  
9 processing is performed for both the source and  
10 destination end point pairs. The processing for the  
11 source pair is the same as that performed when a FIN is  
12 received on an unspliced flow. The processing for the  
13 destination pair is the same as what takes place when a  
14 FIN is sent on an unspliced flow. Finally, the segment  
15 is passed to the IP output routine.

16 As described above, the preferred embodiment of TCP  
17 flow splicing recomputes the TCP checksum for each  
18 packet forwarded through a splice after the packet  
19 header has been modified. Techniques in common practice  
20 allow the new TCP checksum to be calculated by  
21 referring to only the original TCP checksum value and  
22 the changes made to the packet. This can improve  
23 performance compared to computing the checksum from  
24 scratch because it eliminates the need to reference all  
25 the data in the packet.

26 Preliminary checks are performed on the TCP segment  
27 in block 1502 of Figure 15. These checks may or may  
28 not include TCP checksum calculation. Performing the  
29 checksum allows corrupt packets to be detected and  
30 eliminated as soon as possible which may save network  
31 bandwidth. Processing requirements are reduced,  
32 however, if the checksum calculation is eliminated from

1 the splice. Elimination of the checksum is feasible as  
2 corrupt packets will still be detected by normal TCP  
3 processing at the destination node.

4 The flow splice may be defined in terms of packet  
5 sequences. Figure 18 depicts an example of a sequence  
6 of packets associated with the scenario depicted in  
7 Figure 8. In this scenario, each of two nodes A 801  
8 and B 803 has established a connection with an  
9 intermediate node I 802. In addition to the elements  
10 explicitly depicted in Figure 8, the scenario  
11 associated with Figure 18 includes a program on node I  
12 802 that continuously reads data from end point pair  
13  $EP_{I1}$  811 and immediately writes any such data on end  
14 point pair  $EP_{I2}$  812. This corresponds to reading data  
15 from flow  $F_{AI}$  804 and writing it to flow  $F_{IB}$  806.

16 The packet flow depicted in Figure 18 results from  
17 node A 801 sending 100 bytes of data on end point pair  
18  $EP_{A1}$  810. These 100 bytes are contained in a single  
19 packet 1801. Subsequent to receiving this packet 1801,  
20 node I 802 sends an acknowledgment packet 1802 to node  
21 A 801. The acknowledgment packet 1802 indicates to  
22 node A 801 that the 100 bytes contained in the first  
23 packet 1801 have successfully arrived at the  
24 destination of the flow on which they were sent, that  
25 is the destination of flow  $F_{AI}$  804 which is node I 802.

26 Also subsequent to the arrival at node I 802 of  
27 packet 1801, node I 802 sends the 100 bytes of data  
28 received from node A 801 on end point pair  $EP_{I2}$  812.  
29 Again the data happens to be conveyed in a single  
30 packet 1803. Subsequent to the arrival at node B 803  
31 of this packet 1803, node B 803 sends an acknowledgment  
32 packet 1804 to node I 802. This acknowledgment packet



1 1804 indicates to node I 802 that the 100 bytes  
2 contained in packet 1803 have successfully arrived at  
3 the destination of the flow on which they were sent,  
4 that is the destination of flow  $F_{TB}$  806 which is node B  
5 803.

6 Although Figure 18 depicts a specific strict  
7 temporal ordering of the transmission and reception of  
8 the four packets 1801, 1802, 1803 and 1804, certain  
9 other orderings are consistent with the depicted  
10 scenario. The only requirements on the temporal  
11 ordering are that each of packets 1802 and 1803 is sent  
12 from node I 802 after packet 1801 arrives at node I  
13 802. Furthermore packet 1804 is sent from node B 803  
14 after packet 1803 arrives at node B 803. Of course a  
15 packet can not arrive at its destination prior to being  
16 sent from its source.

17 The conclusive indication that the packet flow in  
18 Figure 18 is associated with two flows that are not  
19 joined in a splice is that node I 802 sends to node A  
20 801 an acknowledgment 1802 for the 100 bytes it  
21 received from node A 801 in packet 1801 before node I  
22 802 receives from node B 803 an acknowledgment for the  
23 corresponding 100 bytes sent to node B 803 in packet  
24 1803. This order indicates that the acknowledgment  
25 1802 sent by node I 802 is generated by normal TCP  
26 processing and not a flow splice at node I 802.

27 Figure 19 depicts a sequence of packets associated  
28 with the scenario depicted in Figure 9. This scenario  
29 is identical to the scenario shown in Figure 8 except  
30 that in Figure 9 flow  $F_{AI}$  804 inbound to node I 802 has  
31 been spliced to flow  $F_{TB}$  806 outbound from node I 802

1 thus transforming the two flows into a single flow  $F_{AB}$   
2 904.

3 Like the packet sequence depicted in figure 18 the  
4 flow shown in 19 results from node A 801 sending 100  
5 bytes of data on end point pair  $EP_{AI}$  907. Again, these  
6 100 bytes are contained in a single packet 1901.  
7 Subsequent to receiving this packet 1901, node I 802  
8 forwards the packet to node B 803 after modifying the  
9 packet headers to make it appear to node B 803 as if  
10 the packet had been sent from node I 802 on flow  $F_{IB}$   
11 806. Unlike the packet exchange shown in Figure 18 the  
12 exchange in Figure 19 does not include an  
13 acknowledgment from node I 802 to node A 801 in  
14 response to data packet 1901 from node A 801 arriving  
15 at node I 802.

16 Like Figure 18, Figure 19 shows node B 803  
17 generating an acknowledgment packet 1903 in response to  
18 the arrival of a packet 1902 from node I 802. As  
19 described earlier the splice mechanism creates packets  
20 containing data forwarded from node A 801 through node  
21 I 802 to node B 803 on flow  $F_{AB}$  904 that are acceptable  
22 to node B 803 as packets that originated at node I 802  
23 and were sent to node B 803 via flow  $F_{IB}$  806. When the  
24 acknowledgment packet 1903 arrives at node I 802, node  
25 I 802 modifies the packet to make it appear to node A  
26 801 that the packet originated at node I 802 and is  
27 associated with flow  $F_{AI}$  804. Node I 802 then forwards  
28 the modified packet 1904 to node A 801.

29 The temporal order of the packets in Figure 19 is  
30 the only one consistent with a spliced flow. That is,  
31 for a spliced flow packet 1904 is not sent from node I  
32 802 before packet 1903 arrives at node I 802, and

1 packet 1902 is not sent from node I 802 before packet  
2 1901 arrives at node I 802. Packet 1903 is not sent  
3 from node B 803 before packet 1902 arrives at node B  
4 803 simply to conform with the TCP protocol.

5 By itself this temporal ordering is not sufficient  
6 proof of a flow splice as it is also possible for this  
7 order to result without a splice. The presence of a  
8 flow splice is conclusively indicated by *causality*  
9 between packets received by and sent by the  
10 intermediate node I 802 on which the splice is  
11 performed. Specifically, in the scenario with a flow  
12 splice depicted in Figure 19, the sending of  
13 acknowledgment packet 1904 by node I 802 is caused by  
14 the arrival of acknowledgment packet 1903 from node B  
15 803 and not by the arrival of data packet 1901 from  
16 node A 801.

17 This causality is refuted if node I 802 sends an  
18 acknowledgment to node A 801 for data packet 1901  
19 before an acknowledgment is received at node I 802 for  
20 the corresponding data packet 1902 sent from node I 802  
21 to node B 803. The causality indicative of a flow  
22 splice is confirmed if node I 802 fails to send an  
23 acknowledgment to node A 801 for packet 1901 if it does  
24 not receive the acknowledgment packet 1903 from node B  
25 803 for the corresponding packet 1902.

26 The causality indicative of a flow splice can also  
27 be confirmed a different way. In the scenario without  
28 a flow splice depicted in Figure 18 the ACK number in  
29 packet 1802 is set by node I 802 to one more than the  
30 greatest sequence number I 802 has received from A 801  
31 on flow F<sub>AI</sub> 804 as mandated by the TCP protocol. In  
32 the case depicted in Figure 19, node I 802 received

1 from A 801 100 bytes starting at sequence number 1000  
2 in packet 1801. Packet 1801 contains sequence numbers  
3 1000 through 1099. Node I 802 therefore sets the ACK  
4 number in packet 1802 to 1100.

5 In the scenario with a splice depicted in Figure 19  
6 the ACK number in packet 1904 is set by node I 802 to  
7 the ACK number in the corresponding packet 1903 minus  
8 the *sequence number delta* value for the splice  
9 following the algorithm for splice processing. The  
10 sequence number delta value for a splice is defined as  
11 the sequence number of a byte sent by the intermediate  
12 node to the splice destination minus the sequence  
13 number of the corresponding byte when received from the  
14 splice source. For example the first of the 100 bytes  
15 in packet 1902, sent from node I 802 to node B 803, has  
16 sequence number 2000. The corresponding byte was sent  
17 from node A 801 to node I 802 in packet 1901 with  
18 sequence number 1000. This the sequence number delta  
19 for this splice is therefore 2000 minus 1000 or 1000.  
20 Thus when node I 802 receives acknowledgment packet  
21 1903 from node B 803 it subtracts 1000 from the ACK  
22 number 2100 and sets the ACK number in the forwarded  
23 packet 1904 to 1100.

24 Although the ACK number in each of packets 1802 and  
25 1904 are set according to distinct algorithms, both  
26 algorithms produced the same value 1100. The flow  
27 splice processing is, in fact, required to produce the  
28 same ACK numbers as produced by normal TCP processing  
29 if it is to remain transparent to the splice source and  
30 destination. Although under normal operation the two  
31 algorithms produce the same ACK number, the specific  
32 algorithm in use at a particular intermediate node can

1 be determined by modifying the acknowledgment packets  
2 sent to the intermediate node.

3 For example to determine if a flow splice has been  
4 created at node I 802, we simply add some amount to the  
5 ACK number in the acknowledgment packet 1903 sent by  
6 node B 803. If a flow splice is not present this  
7 modification will have no effect on the ACK number in  
8 the acknowledgment packet 1904 sent by the intermediate  
9 node I 802. If a flow splice is present, however, the  
10 ACK number in the acknowledgment packet 1904 sent by  
11 the intermediate node I 802 will reflect the  
12 modification to the ACK number in the acknowledgment  
13 packet sent by the destination node B 803.

14 If 1000 is added to the ACK number in the  
15 acknowledgment packet 1903 sent by the destination node  
16 B 803, the presence of a flow splice is indicated if  
17 the ACK number in the corresponding acknowledgment  
18 packet 1904 sent by the intermediate node I 802 is also  
19 1000 more than would be expected according to the TCP  
20 algorithm. It is certain the TCP algorithm did not  
21 generate the acknowledgment packet sent by the  
22 intermediate node I 802 if the ACK number in the packet  
23 corresponds to data not yet received by the  
24 intermediate node I 802.

25 It is noted that this invention may be used for many  
26 applications. Although the description is made for  
27 particular arrangements and applications, the intent  
28 and concept of the invention is suitable and applicable  
29 to other arrangements and applications. It will be  
30 clear to those skilled in the art that other  
31 modifications to the disclosed embodiments can be

1 effected without departing from the spirit and scope of  
2 the invention.  
3 .

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207  
2208  
2209  
2210  
2211  
2212  
2213  
2214  
2215  
2216  
2217  
2218  
2219  
2220  
2221  
2222  
2223  
2224  
2225  
2226  
2227  
2228  
22